Starting Out With Haskell

4 Essential Free Tools

1. Haskell Platform

• The Haskell platform bundles all the fundamental tools you'll need to run Haskell programs.
• Head over to haskell.org/platform to download the proper version for your operating system.
• Don't worry if you don't understand all the different components at first!

The Haskell Platform includes:

I. GHC (Glasgow Haskell Compiler) -

- The compiler is responsible for turning your Haskell source code into executable programs.
- Obviously this is the most important thing you need!

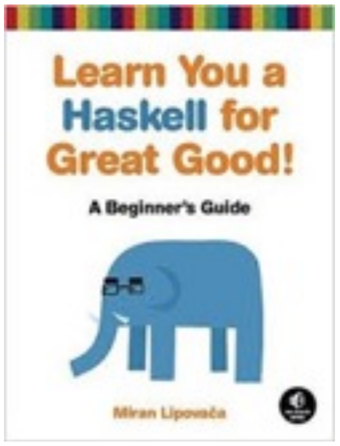
II. Cabal

- Haskell has a lot of open source libraries available for you to download and use!
- The Cabal build system allows you to organize your project so that it will automatically download libraries that you need.

III. Stack tool

- Built on top of Cabal, Stack further refines the package organization and download process.
- Stack uses groups of libraries which are known to work well together, so that you do not have conflicting requirements.
- It makes sandbox-builds the default option, so that it is easier for you to have multiple Haskell projects on the same computer.

2. Learn You a Haskell for Great Good!



- If you're totally new to Haskell, Learn *You a Haskell for Great Good!,* by Miran Lipovaca is a great place to learn some of the basics.
- It cover a lot of ground in a fun way with lots of cute pictures, and it is quite easy to read.
- The online version is freely available at learnyouahaskell.com.

3. H-Lint

- There can be a lot of different ways to write effectively identical code in Haskell.
- The hlint tool helps you learn coding style by pointing out areas where your code can be simplified.
- It points out redundant code, such as unnecessary parentheses.
- It can also show how to simplify your code using library functions.

```
./src/Lib.hs:8:12: Suggestion: Redundant bracket
Found:
  print $ (concat (map toUpperString myWords))
Why not:
  print $ concat (map toUpperString myWords)

./src/Lib.hs:8:21: Warning: Use concatMap
Found:
  concat (map toUpperString myWords)
Why not:
  concatMap toUpperString myWords
```

It is a must have for Haskell developers, both new and experienced. You can install hlint on your system using Cabal with the following commands:

```
>> cabal update
>> cabal install hlint
```

Then you simply navigate to the directory with your source code and use the command:

```
>> hlint .
```

4. H-Pack

- When Cabal builds your Haskell project, it uses a file (typically called {yourproject}.cabal) to determine what libraries it needs to download, and which pieces of your project depend on which libraries
- It also should list all the different modules of your project
- When you add a new library dependency to a module, you'll have to add that as a dependency in the .cabal file, often in a couple different places!
- The hpack library solves a lot of the problems with .cabal files.
- Instead of editing the .cabal file directly, you'll edit a file called package.yaml, with a simpler language in a format that is easier to understand.
- It handles most of the redundancy so that you don't have to!

To install hpack, simply use the command:

```
stack install hpack
```

Then run the "hpack" command in your project directory containing a package.yaml file, and it will automatically generate your .cabal file!

Starting Out with Stack and Hpack

- As a bonus, let's go through the basics of using these tools.
- To make your first haskell project, start with the command:

```
>> stack new MyFirstProject
```

- This might take a few seconds to download a template, thus it requires an internet connection.
- After that, you can look around the new "MyFirstProject" directory and you'll see MyFirstProject.cabal.
- But if you want to instead use Hpack to generate your .cabal files, create this as your package.yaml file:

```yaml
name: MyFirstProject
version: 0.1.0.0
dependencies:
  - base

library:
  source-dirs: src/

executables:
  MyFirstProject-exe:
    main: Main.hs
    source-dirs: app/
    dependencies:
      MyFirstProject

tests:
  MyFirstProject-test:
    main: Spec.hs
    source-dirs: test/
    dependencies:
      MyFirstProject
```

- The package.yaml file says that your project has a library, located in the “src” folder, an executable, defined in “app/Main.hs”, and a test suite, defined in “test/Spec.hs”.
- The executable and test suite depend on your library (“MyFirstProject”), and everything depends on the “base” library.
- Once you have this package file, you can run the “hpack” command and it will generate a .cabal file very similar to the original one created.

Stack Commands

Once you have your project created, you should know the following stack commands:

```
>> stack setup
```

- The setup command determines which compiler version to use and downloads it if necessary.
- Unless you specify otherwise, it will just use whichever version of GHC you installed system wide.

```
>> stack build
```

- The build command compiles your library and executable(s).
- It also creates the executable files beneath the project specific .stack-work directory.

```
>> stack install
```

- The install command copies the executable(s) created from building to your local path.
- After running this, you can run the executables from anywhere on your system.

```
>> stack exec MyFirstProject-exe
```

- Instead of using the install command, you can use the exec command to run a particular executable.
- Since your project can have multiple executables, you have to provide the name of the executable you want.

```
>> stack test
```

- The test command compiles your project and runs the test suites specified in your package.yaml or .cabal file.
- Test suites are essentially executables designed to check your code correctness.

```
>> stack ghci
```

- The ghci command will launch a GHCI prompt with all your code loaded.
- This lets you quickly test some of your functions to see if they work how you expect, without having to modify your executable code.